(2)

THE APPLICATION OF REGRESSION-BASED AND FUNCTION POINT SOFTWARE SIZING TECHNIQUES TO AIR FORCE PROGRAMS

THESIS

Frank Albanese, Jr.
Captain, USAF

AFIT/GCA/LSY/88S-1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

(2)

AFIT/GCA/LSY/88S-1

THE APPLICATION OF REGRESSION-BASED AND FUNCTION POINT SOFTWARE SIZING TECHNIQUES TO AIR FORCE PROGRAMS

THESIS

Frank Albanese, Jr.
Captain, USAF

AFIT/GCA/LSY/88S-1

Approved for public release; distribution unlimited

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information is contained therein. Furthermore, the views expressed in the document are those of the author and do not necessarily reflect the views of the School of Systems and Logistics, the Air University, the United States Air Force, or the Department of Defense.

AFIT/GCA/LSY/88S-1

THE APPLICATION OF REGRESSION-BASED AND FUNCTION POINT SOFTWARE SIZING TECHNIQUES TO AIR FORCE PROGRAMS

THESIS

Presented to the Faculty of the School of Systems and Logistics
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Cost Analysis

Frank Albanese, Jr., M.A.
Captain, USAF

September 1988

Acknowledgements

First, I would like to thank my advisor, Mr. Daniel V. Ferens, and my reader, Dr. Roland D. Kankey, for their dedicated efforts in helping me complete this thesis.

I would also like to thank class GCA-88S for their advice and support throughout the whole thesis process.

Finally, a special thanks to my wife Norma and daughter Kristen Nicole. Without their support and understanding through these last few months, I would have never been able to complete this study.

Frank Albanese, Jr.

Table of Contents

List of Tables

Abstract

This study investigated the ability to develop regression models to predict the number of source lines of code (SLOC) and the degree of correlation between function points and the number of SLOC. Since software size -- or the number of lines of code (LOC) -- is considered to be the primary software cost driver, accurate software size estimates are critical because of the growing importance of software in today's Department of Defense (DoD) weapon systems.

For the regression analysis, four sizing data bases, containing various functional (independent) variables, were used. These variables included complexity, reliability, the program's quality of specification, etc. Regression analysis for each data base was performed with the goal of deriving an optimal model to predict SLOC. Linear forms and nonlinear transforms of the independent variables were used in the analysis. Of the three data bases containing complexity, the squared exponential transform of the variable was statistically significant in two. The best model found was for the Ballistic Missile Office (BMO) data base. The coefficient of correlation was .9109 for the variables INPTLOG (the log of the number of program inputs) and OUTPTLOG (the log of the number of program outputs).

For the function point analysis, the variables of number of inputs, outputs, and interfaces (from the BMO data base) were used to apply Albrecht and Gaffney's methodology to investigate the correlation of function points to software size. No correlation for this particular data base could be found. More data is needed before further research can be accomplished.

THE APPLICATION OF REGRESSION-BASED AND FUNCTION POINT SOFTWARE SIZING TECHNIQUES TO AIR FORCE PROGRAMS

I. Introduction

This chapter provides a summary of the research. First, the issue of rising software costs and its importance to the Department of Defense (DoD) is explained. Second, the specific problem of estimating the size of a software development effort is addressed. Next, the scope of the study is described and pertinent terms are defined. Finally, the specific research questions which this thesis will answer are cited.

General Issue

The increasing complexity of newly developed weapon systems is partially a function of the computer technology embedded in their subsystems. That is,

> Virtually every system in the current and planned inventory makes extensive use of computer technology. Computers control the targeting and flight of missiles, coordinate and control sophisticated systems within high performance aircraft, and integrate the complex activities of battlefield command. Consequently, software has become the dominant factor in military systems [20:52].

The Army Science Board in a 1983 study dealing with acquiring Army software also noted that

> The inventory of mission critical, embedded computers within the DoD is expected to grow from approximately 10,000 in 1980 to over 250,000 by the end of the decade. It is important to recognize that software will assume an ever-increasing role in these systems. Software is no longer merely one part of a system. It is the integration function for the entire system, whether the system is avionics, missile, or command and control functions [33:Sec 2,4].

Software "has become the dominant factor" in new DoD weapon systems because over the past two decades, computer hardware costs have rapidly decreased, while the cost of developing the software to support this hardware has rapidly increased (18:1). Rather, president of a software consulting firm, supports this observation by stating:

> During the thirty-year life span of the computer industry, the cost of computer hardware and related equipment has dropped steeply, while its power has increased beyond the most optimistic projections. Paradoxically, in spite of this inexpensive powerful hardware, software costs have escalated wildly, and represent a large and growing element in most aerospace and defense R&D budgets [25:37].

The increasing importance and cost of software development efforts combined with the recent cost overruns encountered by DoD development projects have piqued the interest of Congress in development projects' cost estimates. That is, when Congress decides to allocate billions of dollars for new weapon systems, the cost estimates for these weapon systems become extremely important. As noted by the 24 May 1984 Comptroller General's Report to Congress, "The accuracy, completeness, and timeliness of DoD's cost estimates need to be improved to give Congress more reliable data for its decision process" (30:1).

Since software costs make up a large portion of a new weapon system's development cost, a key component of accurate, complete, and timely weapon system cost estimates is accurate software cost estimates. Moreover, since software size -- or the number of lines of code (LOC) -- is considered to be the primary cost driver of a software development effort, an accurate estimate of the software package's size becomes extremely important. However, Whetstone notes that

> . . . the size of the software has been consistently underestimated, thus lowering the software cost estimate and, consequently, the overall cost estimate for the weapon system [32:2].

The importance of software size estimates and an accurate sizing methodology are discussed by Wheaton in her 1986 article:

> Software size estimates almost always grow over the life cycle. The amount of underestimation varies depending on many factors; but, on the average, it is in the range of 70 to 100 percent from contract award to project completion. For this reason, it is imperative that greater efforts are applied towards obtaining more accurate size estimates earlier in the software life cycle [31:17].

Research Objectives

Because defense funds are being reduced, it is essential that cost estimates of future weapon systems be as accurate as possible. A key element in ensuring this is to provide more accurate predictions of software packages' sizes. According to a paper presented at the 19th Annual DoD Cost Analysis Symposium, the authors identified (through review and survey)

". . . four types of sizing methods as being representative of the general methods used in DoD application" (14:3). The four sizing methods cited in the paper are PERT sizing, qualitative functional relationships, quantitative functional relationships, and measurement techniques. Although the authors noted that quantitative functional relationships provided only "moderately accurate" software size estimates, they recommended that "research should continue to investigate statistical relationships to estimate the size for specific software functions" (14:5). Moreover, while Whetstone's research failed to develop a "valid" software sizing model using regression techniques that incorporated linear forms of the size drivers, it may be possible to develop a "valid" equation or sizing model by using regression techniques that consider both linear and nonlinear forms of the size drivers (Whetstone's recommendation #2).

Thus, the primary objective of this study is to develop equations using regression techniques (which consider both linear and nonlinear forms of the size drivers) that relate the software's size to the software's functional characteristics (complexity, language type, etc.). A secondary objective is to investigate (where the data allows) the ". . . degree of correlation between 'function points' and the eventual 'SLOC' (source lines of code) of the program . . ." (1:639).

Scope

This study attempts to develop equations that relate functional characteristics of the software to the number of source lines of code (quantitative functional relationship) for the four data bases used. Three of the four data bases were extracted from Whetstone's study and the fourth was obtained from the Electronic Systems Division's Cost Analysis Office (ESD/ACC), Air Force Systems Command (AFSC). While Whetstone's research limited itself to regression techniques involving only linear forms of the independent variables, this study will develop size estimating equations based upon nonlinear as well as linear forms of the independent variables. Moreover, this study will investigate the ability of Albrecht and Gaffney's function point methodology to predict the number of source lines of code for the applicable data.

Definitions

The following terms are crucial to this study:

Computer Software (or software).

> A combination of associated computer programs and computer data required to enable the computer hardware to perform computational or control functions. Computer software includes that computer data defined within and integral to a computer program but typically does not include variable data values, such as mission data or test data, that may be entered into the software immediately preceding software execution [33:Sec G,3].

Computer Software Configuration Item (CSCI). "These are programs, or group of programs, which satisfy common functions and are managed as separate entities" (8:3).

Cost Driver. "A characteristic of a system or end item that has a large or major effect on the system's cost" (2:Sec A,21).

Function Points. ". . . essentially, a weighted sum of the numbers of 'inputs,' 'outputs,' 'master files,' and 'inquiries' provided to, or generated by, the software" (1:639).

Multiple Regression. " . . . a statistical tool that utilizes the relation between two or more quantitative variables so that one variable can be predicted from the others" (22:23).

Size Driver. Similar to a cost driver. A functional characteristic of the software that has a major effect on software size.

SLOC. "An instruction written in assembler or HOL (higher order language) is often referred to as a source line of code (SLOC) to differentiate it from a machine instruction" (8:3).

Source Instructions.

> This term includes all program instructions created by project personnel and processed into machine code by some combination of preprocessors, compilers, and assemblers. It excludes comment cards and unmodified utility software. It includes job control language, format statements, and data declarations. Instructions are defined as lines of code or card images [4:59].

Research Questions

The two research questions addressed by the thesis are:

1. Using size drivers identified in the literature and identified by Whetstone's study, how well can regression analysis relate software size to these size drivers?

2. Given the applicable data, what is the degree of correlation between function points and the actual size (SLOC) of the software?

Research Development

The research conducted in this thesis will follow the chapters outlined below.

Chapter one introduces the research. It discusses the general issue of software costing and sizing, the research objectives, the scope of the research, the definitions that are crucial to the study, and the research questions to be answered.

Chapter two cites the literature pertinent to the area of software sizing.

Chapter three discusses the methodology used to conduct the study.

Chapter four contains the results of the statistical analysis and function point investigation.

Finally, chapter five discusses the conclusions reached from conducting this research and recommends areas for further study.

II. Background

Overview

This chapter presents a literature review of relevant software costing issues and the prominent software sizing techniques. First, software cost trends and the importance of cost estimating software will be explained so that a better understanding of how the software size estimate fits into the overall software cost estimate may be obtained. Moreover, those characteristics (attributes) of the software that have a significant influence on development costs will be cited here. Second, the primary cost driver of a software development effort, software size, will be discussed. Then, the major software sizing techniques in use today will be described (including Albrecht and Gaffney's function point methodology). Finally, the review will conclude with the author's comments.

Software Cost Trends

As software costs increase, the importance of the software cost estimating process will also increase. Jensen and Lucas describe software cost trends in the United States.

> The annual cost of software in the United States in 1980 was approximately $40 billion, or about 2 percent of the gross national product. The Department Of Defense (DoD) software budget in 1980 was $3 billion. The rate of software growth is considerably greater than that of the United States economy in general. In 1982 the DoD software budget exceeded $5.5 billion dollars, and the DoD projection for 1990 is in excess of $32 billion [17:1].

As Steig notes in his thesis, this $32 billion of the DoD's budget will account for " . . . 85% of the budget spent on computer related acquisitions . . ." (29:6). Moreover, this figure ($32 billion) will also represent between 15% and 20% of the DoD's total budget (20:53). Boehm projects that

> Using a 12% annual growth rate, the annual U.S. software cost would be roughly $70 billion in 1985 and $125 billion in 1990. Comparable world software costs are difficult to calculate due to differing salary scales, but they would be at least twice this high: over $140 billion in 1985 and over $250 billion in 1990. Clearly, these costs are sufficiently large to merit serious efforts to understand and control them [5:33].

Although software's increasing importance in defense system applications has been recognized, rampant cost overruns associated with software projects are not unusual. Steig notes:

> Cost model developers have concluded that despite the progress made to date, a need for more accurate cost estimating results continues to exist within the software industry. The state-of-the-art of software cost estimating is only mediocre. The best software cost estimating models are only within 20% of the actual, and they do that in only 70% of their predictions [29:7].

Why are software development efforts difficult to predict? In a course handout entitled, "Software Engineering Project Management," Computer Economics, Inc. cites the following reasons why software efforts are difficult to estimate:

a) Over optimism
b) Difficulty of scoping the job
c) Conflicting goals
d) Conflicting definitions
e) End products may not be seen for a long time [11:Sec 1,7].

Moreover, Barry Boehm relates that a person

> . . . can't estimate the cost of producing 100,000 source instructions of software as accurately as we can estimate the cost of producing 100,000 aspirin tablets . . . [4:32].

The reasons for this are:

> Source instructions are not a uniform commodity, nor are they the essence of the desired product.
> Software requires the creativity and cooperation of human beings, whose individual and group behavior are generally hard to predict.
> Software has a much smaller base of relevant quantitative historical experience, and it is hard to add to the base by performing small controlled experiments [4:32].

Thus, in order to better recognize and control these estimating problems, an understanding of the importance of the software cost estimating process is needed.

Estimating Software Cost

The difficulty and importance of the software cost estimating process is noted by Jensen. He states that

> Software development has been characterized by severe schedule slippage, cost overrun and the inability of the developer to estimate the resources and schedule required early in the requirements analysis and functional design phase when critical investment decisions must be made. This estimation difficulty has emerged as one of today's most critical development problems [16:1].

Boehm also attests to the importance of software cost estimation by stating that software cost estimates "provide the vital link between the general concepts and techniques of economic analysis and the particular world of software engineering" (4:30). Software cost estimating helps determine the cost effectiveness of a software development effort. Moreover, Boehm describes the problems that software projects

experience without an accurate cost-estimation capability:

> 1. Software project personnel have no firm basis for telling a manager, customer or salesperson that their proposed budget and schedule are unrealistic. This leads to . . . the inevitable overrun and performance compromises as a consequence.
> 2. Software analysts have no firm basis for making realistic hardware-software tradeoff analyses during the systems design phase. This often leads to a design in which the hardware cost is decreased at the expense of an even larger increase in the software cost.
> 3. Project managers have no firm basis for determining how much time and effort each software phase and activity should take. This leaves managers with no way to tell whether or not the software is proceeding according to plan [4:30].

As Whetstone notes, "because of these factors, the cost analyst will constantly be faced with a challenge to accurately estimate the costs of software" (32:11).

Understanding the importance of the software estimating process is necessary. However, it is equally important to recognize the cost drivers of a software development effort. For example, the COCOMO parametric cost model recognizes thousands of delivered source instructions (KDSI) along with 16 software attributes (partitioned into the four categories of product, computer, personnel, and project) as the main drivers of cost. Another parametric model, the RCA PRICE-S model, uses SLOC, software application, resources, and software complexity as the primary inputs (cost drivers) to the model (8:12).

In a thesis completed at the Naval Postgraduate School, Park cites the following about cost attribute factors:

> Stanley [Ref. 10] defines two major areas of software cost drivers: project specific factors and organization

dependent factors. Boehm [Ref. 9] identifies five factors which closely match Stanley's. Size attributes, program attributes and computer attributes fall into Stanley's project specific factors. Personnel attributes and project attributes fall into Stanley's organization dependent factors. Wolverton [Ref. 12] suggests that top-level characteristics are parameters which can be classified into software structural parameters and project financial parameters. Software structural parameters may be divided into size, program attributes, hardware attributes, project attributes, and environmental attributes. Project financial parameters are divided into direct labor charges, overhead, other direct charges, general and administrative expense, and fee. Bruce and Pederson [Ref. 13] divide cost drivers into four categories: requirement factors, product factors, process factors and resource factors [23:24-25].

In a Rand study prepared for the United States Air Force, the author groups the input parameters (used as the independent or explanatory variables in a software cost model) into three characteristic classes:

> . . . of the software itself, of the environment within which the software is used or was developed, and associated with the functions that the software performs. Examples of each follow.

Software Characteristics:
- Size (number of instructions)
- Type (operating, support, applications)
- Application category (business, scientific, avionics)
- Complexity/Difficulty
- Quality
- Language

Environmental Characteristics:
- Operation Environment
  - Hardware characteristics
    - Memory size
    - Speed
  - System Architecture
- Development Environment
  - Personnel
    - Qualification
    - Experience with language
    - Experience with hardware
  - Development process

Development time
Programming practices
Stability of design/requirements
Concurrent hardware development
Target and host computer not the same
Access to development computer
Extent of documentation

Functional Characteristics:
Number of targets tracked
Number of flight modes [19:14-15].

Although different studies and estimating models may cite and use different attributes of a software development effort as cost drivers, there still exist many similarities between lists of cost drivers. Moreover, most software cost models in use today recognize software size or the number of source lines of code (SLOC) as the primary cost driver of software development.

Software Size

Many sources cite software size as the primary driver of cost for a software development effort. Joseph Fox notes that the size of the software package is the most important driver of development costs. In his book, Software and it Development, Fox explains that "the difficulty of software development rises nonlinearly with the size of the program to be written" (9:242). Garvey and Powell from Mitre Corporation state that "the size of the software system is typically the critical source of uncertainty, and has historically been the most significant driver of cost" (12:76). In his thesis, Park cites the following:

> In software estimating models, software size is the key parameter influencing cost. All the proven estimating techniques begin with an estimate of the size of the software package and then at various levels of sophistication produce an estimate of cost or time based on size and calculated or derived productivity factors [23:10].

In a paper written for The Journal of Parametrics, Richard Reese and Jim Tamulevicz state:

> The most popular measure of software size is the number of lines of code. The estimation of the number of lines of code is important since most cost estimating tools base their projected estimate upon this number. There are many other parameters used in conjunction with various cost estimating tools including complexity, personnel capabilities, and reliability requirements of the system to name a few. However, the number of lines of source code is the most important factor. A poor lines of code estimate can result in a bad estimate of of the total project effort [26:35].

Boehm also recognizes the importance of software size:

> The biggest difficulty in using today's algorithmic software cost models is the problem of providing sound sizing estimates. Virtually every model requires an estimate of the number of source or object instructions to be developed, and this is an extremely difficult quantity to determine in advance [3:17].

While software size is recognized as the most important cost driver of software development, defining size and developing sizing techniques are not "clear cut" tasks. Software size is usually considered a "measure of the magnitude of a software product with common units of measure being lines of source code" (21:A-31520).

Lines of source code (or SLOC) as a measure of size is preferred (by most software cost modelers) to the number of machine instructions because

> . . . the number of machine language instructions is affected by the function of the language and the efficiency of the compiler. If compensation is not made for these factors it is possible for a single source program to be responsible for different numbers of machine language instructions simply because a different compiler is used [26:36].

Wheaton states the same reasons for not using machine language instructions to estimate software size.

> Some of the current models define the effort/size relationship on the basis of object or machine language instructions (MLI). The use of MLI for estimating the size of software is not recommended, because it is best to consider lines of code as units of effort which comprise the total software development effort. This is not possible with MLI as they are a function of the language and compiler efficiency, and not directly related to effort. Using MLI does not provide a consistent basis for measuring effort, since the same source program may generate different numbers of object instructions depending on the compiler [31:17].

However, the term "size" in many sizing efforts does not carry the same meaning. As Dekker and Bosch note in their study,

> First of all one can measure size in terms of object instructions or in terms of source instructions. Secondly, it is not always clear which instructions should be included when measuring size. Some authors count only delivered instructions, others count test software too, which is not delivered [6:6].

Graver et al. discuss the types of code to be considered when measuring the size of a software project:

> Throwaway code--code written and used for the development of the delivered programs, but not itself delivered.
>
> Converted code--code taken from earlier programs but adapted in detail for this development.
>
> Off-the-shelf code--code taken from earlier programs without adaptation.

Computer data--code that supplies values for variables in the program (such as DATA statements in FORTRAN programs).

Comment statements--code intended to be read by people and not the computer [13:14].

Developing a sizing technique can be as difficult as defining software size. Why is software sizing difficult? Because, it must be done early in the development phase, when "requirements are neither firm nor fully defined, and the system architecture is volatile" (28:3). Moreover, "past performance data is often lacking and past experience may not be applicable" (28:3).

Software Sizing Techniques

Since the objective of this study is to develop software sizing equations, it is appropriate to describe some of the existing sizing techniques in use today.

Reese and Tamulevicz examine several sizing techniques which include:

a) PERT sizing
b) Albrecht's Function Points
c) Data Base Analogy
d) Parametric Sizing Tools [26:38].

However, Ferens, in a software sizing paper presented at the 1988 National Aerospace and Electronics Conference (NAECON), notes that ". . . there are various schema for categorizing models such as those used for software sizing" (7:1). Thus, Ferens divides software sizing techniques into five classes (based upon his own research): expert judgement models (e.g., PERT sizing), analogy models, parametric models, function

point models, and regression-based models (7:1). Although Perens notes five categories of sizing techniques, four of the five overlap Reese and Tamulevicz's list of sizing techniques. All five categories of sizing techniques will be briefly described below.

Expert Judgement Models. These are models that are based upon experts' "opinions." That is, "one or more experts are consulted for their ideas regarding the size of a program or factors which affect software size" (7:2). These sizing models can be used as stand-alone models; however, expert judgement techniques are usually used in conjunction with one or more of the other techniques. Perens cites an excellent example: ". . . expert judgement could be used to determine the inputs for a regression-based model" (7:2).

The prominent expert judgement sizing technique is the Program Evaluation and Review Technique (PERT). PERT sizing is based on the assumption that the experts (software engineers) can provide realistic size estimates for new projects based on prior experience. "PERT sizing uses a statistical approach to estimate the size of a program" (26:38). The development effort is first broken down into components (separate blocks of code based on the function performed). Then, a most likely size estimate as well as upper and lower bound estimates and standard deviations are calculated for each component. The three values are then "averaged" for each component. Finally, the component estimates are

summed to obtain a total estimate (26:39). The equations used in this technique are

$$E(i) = [a(i) + 4m(i) + b(i)]/6 \quad (1)$$

$$S(i) = [b(i) - a(i)]/6 \quad (2)$$

where

E(i) = the estimated size of the software component
a(i) = the lowest possible size of the software component
m(i) = the most likely size of the component
b(i) = the highest possible size of the component
S(i) = the standard deviation [26:39]

As Reese and Tamulevicz note, "a basic problem with PERT sizing is the assumption that the estimates are unbiased toward either underestimation or overestimation" (26:39). However, in reality, ". . . the m(i)'s cluster toward the lower limit resulting in an underestimation" (26:39).

Another expert judgement technique described by Ferens is the Software Sizing Model (SSM) developed by Bozoki. SSM is a statistical method that generates component sizes, total project size, and standard deviations. SSM inputs (each requiring expert opinion) are: pairwise data, ranking data, sorting data, and PERT sizing data (7:2). Reese and Tamulevicz describe the SSM:

> The relative sizes of the components are estimated based upon four input data sets. These data sets include information describing pairwise, ranking, sorting, and PERT sizing data. The pairwise data consists of the selection of the larger of two modules as solicited by SSM. The ranking process simply asks the user to rank the modules by size. Sorting involves the assignment of

> each module to a specific size range as provided by SSM. PERT sizing uses the PERT sizing technique . . . [26:44].

The primary advantage of expert judgement models is that they usually do not rely on historical data, thus, making these models ". . . useful for new programs for which historical precedents do not exist" (7:3). Moreover, these models are also useful early in program development when data is scarce.

However, Ferens notes that these models do have drawbacks:

> Expert judgement models are highly dependent on personal opinions, which may be subjective and biased. Also, finding a true "expert" may be difficult; even the assessments of knowledgeable personnel are sometimes mere guesses [7:3].

Data Base Analogy Models. A different approach to the sizing problem involves breaking down a software development project into components (a functional group of code) and then comparing these components with components of a similar existing software product.

Ferens describes a simple equation for analogy estimation:

$$S = F \times (\text{Size of Similar Packages}) \qquad (3)$$

"where 'S' is size (usually in SLOC) and 'F' is a factor determined by experience or politics" (7:1). While Ferens notes that the "F" factor is difficult to determine, Reese and Tamulevicz consider three attributes of a software

component's function which would facilitate the comparison process (facilitate the determination of the "F" factor described by Ferens). The three attributes of the component's function are: complexity, application environment, and the extensiveness of project requirements (26:41). Reese and Tamulevicz discuss these attributes:

> The difference in complexity can result in a significant variation in amount of effort. Complexity issues include the accuracy or precision of the outputs, the amount of complexity within and without the system, and the required reliability of the system.
>
> Differences in development and application environments will affect the size of the software. For example, the number and quality of development tools may differ resulting in a disparity of effort required. Also, the inherent problems of mobile applications are different from that of a fixed environment.
>
> The project's requirements will have a significant impact when comparing components. There may be differences in the number and types of interfaces, the extent of exception handling required, or the size of the supporting data base. These and other factors need to be considered when estimating size based upon previous work [26:41].

One example of an analogy model is Aerospace Corporation's Software Size Estimator (SSE). According to Ferens, the SSE model contains a historical data base consisting of space system software projects. A user of the model would input software type (spacecraft telemetry, for example) and an assessment of the software's complexity. Then, SSE would search its historical data base for similar programs and compute a "probable size estimate" for the proposed project based on the similar programs found in the data base (7:2).

The advantages in using analogy models are:

> . . . the estimate is based on actual programs for which historical data exists. They are also useful early in a program because minimal input data is required [7:2].

Moreover, analogy models allow "the mapping of functional requirements to module size" and ". . . provides an accurate baseline to size new projects" (26:41).

Although this software sizing technique is very useful, it is not without its disadvantages. This technique is very time consuming. Also, while relying heavily upon historical data is an advantage to this technique, it can also be a disadvantage. That is, historical data may not exist, may be unavailable, may be inaccurate, or it may be incomplete (26:42). Reifer explains that "most experienced engineers and managers in the industry put very little confidence in the sizing estimates developed . . ." by this technique (27:1).

Parametric Models. "These models use input parameters consisting of numerical or descriptive values of selected program attributes" (7:3). The idea of parametric sizing methods is an "offshoot" of software parametric cost estimating techniques (COCOMO or PRICE-S, for example).

One example of a parametric sizing model is the RCA PRICE Sizer (SZ) model discussed by Ferens, Reese, and Tamulevicz. Although there exist two versions of the model -- one for military applications and one for commercial applications -- to reflect differences in the nature of

development environments, "both models incorporate information relating to software design techniques, technical approaches, growth requirements, input/output functions, and historical data" (7:4; 26:46).

Moreover, Reese and Tamulevicz note that the PRICE Sizer model was intended to be used throughout a software development project as a management tool. That is, the model could be used early in the development process when very little information is available, or it could be used later when more information has been obtained (26:46).

While each version of the model requires 15 or more inputs, some of the input parameters are similar to those used by function point methodology. As Reese and Tamulevicz state, "these include the number of output pages, alphanumeric displays, input streams, and output streams" (26:46).

Parametric sizing models' advantages are their objectivity, efficiency, and their ability to be calibrated. However, these sizing models also possess the "inability to handle exceptional situations" (outstanding personnel, new applications, etc,) and the "inability to compensate for poor input" (26:47).

Function Point Analysis. Another software sizing technique is Albrecht and Gaffney's function points. This technique avoids the use of the number of lines of code altogether (this technique assumes that software size based on the number of lines of code does not measure development

difficulty). Function point analysis estimates the number of functions required for program development and "any factor which would make that function more or less difficult to implement" (26:40). That is, the thesis of Albrecht and Gaffney's work is

> . . . the amount of function to be provided by the application (program) can be estimated from an itemization of the major components of data to be used or provided by it. Furthermore, this estimate of function should be correlated to both the amount of "SLOC" to be developed and the development effort needed [1:639].

The approach that this methodology takes is to list and count the number of inputs (I), the number of outputs (O), the number of inquiries (Q), the number of master files (M), and the number of interfaces (N) ". . . to be delivered by the development project" (1:639). Then, based on an equation developed by Behrens reflecting the relative value of each function (to the user), the number of function points is calculated. Behren's formula for the number of function points in a software package is

$$F = 4(I) + 5(O) + 4(Q) + 10(M) + 7(N) \qquad (4)$$

where each letter represents those functions that were described above (1:647). Once the number of function points to be delivered (or used) by the software is determined, this number can then be correlated to SLOC. Albrecht and Gaffney's work with 24 COBOL and PL/1 programs developed the following three equations:

$$S = 118.7(F) - 6,490 \quad (5)$$

$$S = 73.1(F) - 4,600 \quad (6)$$

$$S = 53.2(F) + 12,773 \quad (7)$$

where

S = SLOC
F = the number of function points (1:643)

Eq (5) is the estimating equation that relates the number of function points to COBOL SLOC, Eq (6) relates the number of function points to PL/1 SLOC, and Eq (7) reflects the estimating equation that relates the number of function points to SLOC for the entire data base.

The major advantages in using function points are discussed by Albrecht and Gaffney.

> A major reason for using "function points" as a measure is that the point counts can be developed relatively easily in discussions with the user/customer at an early stage of the development process. They relate directly to user/customer requirements in a way that is more easily understood by the user/customer than "SLOC."
>
> Another major reason is the availability of needed information. Since it is reasonable to expect that a statement of basic requirements includes an itemization of the inputs and outputs to be used and provided by the application (program) from the user's external view, an estimate may be validated early in the development cycle with this approach.
>
> A third reason is that "function points" can be used to develop a general measure of development productivity (e.g., "function points per work-month" or "work-hours per function point"), that may be used to demonstrate productivity trends [1:639].

However, as noted by Ferens, Reese, and Tamulevicz, function point applicability outside of the business/data

processing realm is questionable. Its applicability is highly uncertain with respect to real-time applications, especially command and control systems (although the ASSET-R software cost model attempts to adapt function point analysis to the real-time/scientific environment) (7:3; 26:40). Another problem with function points noted by Reese and Tamulevicz ". . . is the lack of a standard set of definitions for the input values. The terminology used is sometimes unclear" (26:40).

Regression-Based Models. Regression-based sizing models attempt to derive size estimating relationships that relate the size of the software (the dependent variable) to functional characteristics of the software (the independent variables).

An example of a study that used regression analysis to determine a software sizing model is the one conducted by Itakura and Takayanagi. From 38 COBOL programs (batch programs used in a banking system),

> . . . the authors attempted to develop a program size estimation model by looking at the program structure and logic, and determining the number of lines required for each type of process [15:104].

The best fit relationship found by Itakura and Takayanagi was

$$Y = -810 + 310*X(1) + 1.12*X(2) + 553*X(3) + 5.91*X(5) + 1.62*X(7) + 99.7*X(8) \quad (8)$$

where

X(1) = the number of input files
X(2) = the number of input items
X(3) = the number of output files
X(5) = the number of reports
X(7) = the number of vertical items in reports
X(8) = the number of calculating processes (80 steps per one calculating process (15:105,108)

As noted by the authors, ". . . this estimation model is specifically for our projects and cannot directly be adapted to others . . ." (15:109).

Ferens discusses the strengths and weaknesses of regression-based software sizing models.

> Regression-based models share an advantage with analogy models in that they are based on historical data. An additional advantage is that they can be used for programs which do not have directly analogous programs in the historical data base, since the sizing equations transcend the need for direct analogy. They are also useful early in a program if the input factors are available. However, like analogy models, regression-based models are seldom appropriate for programs outside of the scope of the data base from which they were developed. . . . Another difficulty with regression-based models is that a valid regression-based equation should have a high correlation coefficient and a low standard estimating error. Unfortunately, for many data bases, this is not always possible [7:2].

Conclusion

The importance of the software cost estimating process is obvious. Both the size and buying power of the defense budget have shrunk at the same time that software development costs have increased at alarming rates. Thus, better and more accurate methods to predict software development costs are needed to ensure that defense funds are not overallocated

nor underallocated. Ensuring an accurate prediction of software costs appears to hinge on the ability to accurately predict the size of the software itself. As seen from this review, there are many sizing models being used today. However, there are accuracy problems with each technique described in the review. Therefore, the objective of this study is not to develop an "infallible" sizing model, but to hopefully add to existing software sizing knowledge in the area of regression-based and function point sizing models.

III. Methodology

Overview

The methodology that was used to carry out this study will be described here. Initially, a thorough review of Whetstone's thesis and the available software sizing literature revealed that more work needed to be done in the area of regression-based sizing techniques.

Whetstone conducted his study using linear regression techniques. However, Putnam relates the following about software sizing models:

> Morin recommended that ". . . researchers should apply non linear models of data interpretation . . . as Pietrasanta [31] states, the use of non linear methods may not produce simple, quick-to-apply formulas for estimating resources, but estimation of computer resources is not a simple problem of linear cause-effect. It is a complex function of multiple-variable interdependence."
>
> ". . . The most important objective of estimation research should be the production of accurate estimating equations. At this time [1973], the application of non linear methods seems to hold the greatest promise for achieving this objective" [24:345].

Nonlinear models can be classified as intrinsically linear or intrinsically nonlinear. Those models that are intrinsically linear can, through transformations, be expressed in linear form. Those models that are intrinsically nonlinear cannot be so transformed. This study deals with nonlinear models that can be transformed to a linear form.

Using three of the six data bases from Whetstone's study [the Ballistic Missile Office data base, and the two Armament Division data bases (airborne and ground)], a preliminary

analysis of the size drivers was accomplished to help identify SLOC's relationship to the size drivers [individual X-Y plots were inspected to see if they would help reveal the relationships that existed (linear or nonlinear)]. Then, multiple regression analysis was used to develop the statistics of the resulting regression equations. Finally, because it was the only data base that provided the applicable input parameters (the number of interfaces, the number of program inputs, and the number of program generated outputs), the Ballistic Missile Office (BMO) data base was used to investigate the correlation between function points and SLOC.

Data Description

Three of the four data bases used in this study were taken from Whetstone's research. As noted by Whetstone, the ". . . data bases are all different in terms of number of data points and functional description of the software" (32:27).

The first and smallest data base contained data on seven programs that are used on missile systems. These programs were all developed by the Ballistic Missile Office (BMO), Air Force Systems Command (AFSC), and were obtained from HQ AFSC. Each of the seven programs (or software packages) are described by eight functional characteristics: the number of source lines of code (SLOC), the environment in which the program operates (airborne vs. ground-based),

program code type (programming language), the number of interfaces existing between the program and other programs and/or users, the number of program inputs, the number of program generated outputs, programmer experience in months, and the number of months needed to develop the program (32:27-28).

The second and third data bases taken from Whetstone's study were obtained from Armament Division (AD), AFSC. Although the data originally was obtained as one 25-point data base, Whetstone segregated the data into two data bases based on operational environment: a 12-point data base containing all ground-based programs, and a 13-point data base containing all airborne programs. Moreover, both data bases are described by the same functional elements. They are: SLOC, the number of development months, programming language, the quality or degree of system specification (low to high ratings), the reliability expected from the software (low to high ratings), the function supported by the software (missile, range, or munitions), and the software function's degree of complexity (low to high ratings) (32:28-29).

The last data base used in this research is an updated version of the Electronic Systems Division (ESD) data base that Whetstone used. The ESD/MITRE data base (its present name) contains information on 26 software projects, 23 of which have been completed (10:Sec 1,2). Moreover, the data

contained in this data base were captured at the Computer Software Configuration Item (CSCI) level (the three previous data bases described above contain data which were captured at the project level). Although each CSCI is described by 125 different data fields in the ESD/MITRE data base, only five fields were extracted to make up this study's 97-point data base (3:Sec 3,1). The five parameters extracted that describe each CSCI are delivered source lines of code (DSLOC), the complexity of the CSCI's function (very low to extra high ratings), the CSCI's reliability requirement (very low to very high ratings), the use of modern programming practices (MODP -- very low to very high ratings), and the use of software tools (very low to very high ratings). For a more detailed description of these software characteristics and their "effort multiplier" ratings, the reader can consult reference 4.

As noted by Whetstone, ". . . the nonquantitative variables were quantified" (32:30). Thus, since this study is a partial "recreation" of Whetstone's research, this thesis will parallel Whetstone's method of quantifying the nonquantitative independent variables. Whetstone explains:

> The variables of complexity, reliability, and quality of specification were rated . . . for each data base by the organization which assembled the data bases. It has therefore been assumed that the software personnel in each of these organizations were knowledgeable about their own data and have assigned the correct rating to each variable. . . . The harder-to-quantify variable of programming language was quantified by assigning numerical values to each

different language used . . . , per the method used by the ARINC Research Corporation (see 1:4-16) [32;30,32].

Finally, Whetstone quantified the software's operating environment (his method was not explained) and ". . . the function of the system the software operated in . . ." (32:32). Table I reflects the assignment of values determined by Whetstone which this study will use.

TABLE I

Quantification of the Nonquantitative Variables

Complexity Reliability Quality of Spec	Languages	Environment	Function (AD Data Base)
Very Low = 1	Fortran = 1	Ground = 1	Missile = 1
Low = 2	Pascal = 2	Air = 2	Range = 2
Nominal = 3	Assembly = 3	Space = 3	Munition = 3
High = 4	Event Driven Language = 4		
Very High = 5	Jovial = 5		
Extra High = 6	CMS = 6		
	PLM-86 = 7		
	Basic = 8		
	CPL = 9		
	PL/1 = 10		

REPRINTED FROM: (32:31)

Multiple Regression/Function Point Analysis

Multiple Regression. In order to answer this study's first research question (how well can regression analysis relate software size to the software's size drivers?), regression analysis was used to develop a sizing model (a model that predicts SLOC) for each data base. The Statistical Analysis System (SAS) package (version 5.16) was the software used that calculated the regression statistics. The work was accomplished on a VAX 11/785 computer.

First, an analysis of each size driver was accomplished. Each size driver (independent variable) was plotted against the number of SLOC (dependent variable) to reveal any obvious statistical relationships. Neter explains the term "statistical relationships:"

> A statistical relation, unlike a functional relation, is not a perfect one. In general, the observations for a statistical relation do not fall directly on the curve of relationship [22:25].

Next, the SAS package was used to show the strength of the statistical relationships between the size drivers and the number of SLOC for each data base. This was accomplished by separately running each of the independent variables against SLOC and then evaluating each resulting equation's F statistic, R Square (denoted in this study as R^2), and the independent variable's t statistic (each of these regression statistics will be explained later in this chapter).

Then, the SAS package was used to calculate the Analysis of Variance (ANOVA) tables and other regression statistics

for the models being considered. The considered models were identified by stepwise (backward) regression runs. The statistics evaluated for each model were the R^2, the adjusted R^2, the model's F statistic, each of the included independent variables' t statistics, and the coefficient of variation (CV). The two diagnostics considered for each model were multicollinearity and outlying observations (the reader can consult reference 22 for an in-depth discussion of the regression statistics and diagnostics that this study considered).

The regression models were evaluated in the following manner:

1. R^2 (or the coefficient of multiple determination), which measures the proportion of variation in the dependent variable explained by the independent variables, was considered first. Neter explains the danger in considering R^2 alone:

> A large R^2 does not necessarily imply that the fitted model is a useful one. For instance, observations may have been taken at only a few levels of the independent variables. Despite a high R^2 in this case, the fitted model may not be useful because most predictions would require extrapolations outside the region of observations [22:241].

Moreover, the more independent variables included in a model, the higher the R^2 will become. Thus, the adjusted R^2 (adjusted coefficient of multiple determination), which reflects the explanatory value of the additional independent variables, was also considered.

2. The F statistic is used to decide whether or not the hypothesized relationship between the dependent variable Y and the set of independent variables is significant. That is, choosing the null hypothesis that all regression parameters equal zero implies no relationship, and choosing the alternative hypothesis that not all the regression parameters equal zero implies that a relationship exists (22:240-241).

The t test compares each independent variable's t calculated value [t(calc)] with the regression model's associated t distribution to see which variables are statistically significant (22:243). The F test and t test decision rules are described in Table II.

3. The coefficient of variation was considered because it relates the size of the standard error of the estimate to the magnitude of the dependent variable. If estimating from the center of the data, the model's estimate will be within plus or minus two times the coefficient of variation (an approximation only). This study used the guideline of a 30% to 40% coefficient of variation.

4. Since multicollinearity is present in most "nonexperimental" data and has significant effects on other regression statistics, this study evaluated its effects on

TABLE II

F Test/t Test Decision Rules

F test (level of significance at a = .10 for this study):

If F(calc) < or = F[table(1-a;p-1, n-p)], conclude "cannot reject" the null hypothesis

If F(calc) > F[table(1-a;p-1, n-p)], conclude "reject" the null hypothesis

where:

F(calc) = Mean Square Model/Mean Square Error
n = the number of observations
p = the number of parameters in the model

t Test (level of significance at a = .20 for this study):

If |t(calc)| < or = t(1-a/2;n-p), conclude "cannot reject" the null hypothesis (that B(i) = 0)

If |t(calc)| > t(1-a/2;n-p), conclude "reject" the null hypothesis

the developed models. Neter describes some problems that arise when the independent variables of a model are highly correlated:

1. Adding or deleting an independent variable changes the regression coefficients.
2. The extra sum of squares associated with an independent variable varies, depending upon which independent variables already are included in the model.
3. The estimated regression coefficients individually may not be statistically significant even though a definite statistical relation exists between the dependent variable and the set of independent variables [22:382-383].

The SAS package makes available two statistics which help determine the independent variables' degree of correlation. The first statistic is the variance inflation factor (VIF). Neter explains that

> These factors measure how much the variance of the estimated regression coefficients are inflated as compared to when the independent variables are not linearly related [22:391].

Neter further points out that a VIF greater than 10 may indicate that multicollinearity may be influencing the model's predictive ability, and that a major limitation of VIFs ". . . is that they cannot distinguish between several simultaneous multicollinearities" (22:392-393).

The second statistic which evaluates multicollinearity in models (and which is supplied by the SAS package) is the tolerance. The tolerance is expressed as 1/VIF for each independent variable included in the model. Tolerance values will fall in the following range:

$$0 < \text{or} = \text{TOLERANCE} < \text{or} = 1 \qquad (9)$$

Thus, a tolerance value close to zero implies multicollinearity, and a value close to one implies independence. This study used a tolerance limit of .1 or greater for leaving a variable in a model.

5. Finally, the outliers with respect to X and Y and the influential outliers were considered.

Outlying X observations were determined with the help of leverage values (h's). Leverage values indicate

> . . . whether or not the X values for the ith observation are outlying, because it can be shown that h(ii) is a measure of the distance between the X values for the ith observation and the means of the X values for all n observations [22:402].

Thus, ". . . leverage values greater than 2p/n" (where p is the number of parameters in the model and n is the number of X observations) ". . . are considered . . . to indicate outlying observations with regard to the X values" (22:403).

The studentized residual (SRESID) is evaluated to determine outlying Y observations. The SAS package calculates these values for each dependent variable. Then, comparing the SRESIDs with the appropriate t-distribution, the outlying Y observations are identified [that is, if a Y's SRESID is greater than t(1 - significance level, n - p), that Y observation is considered an outlier] (22:405-406).

"After identifying outlying observations with respect to their X values and/or their Y values, the next step is to ascertain whether or not they are influential . . ." (22:407). What is the meaning of influential in this context? If the ith observation is deleted, and the regression coefficients change significantly, then the observation is influential. The SAS package measures the impact of the ith observation on the model's estimated regression coefficients with the

Cook's distance measure (or Cook's D). The guideline used by this study to identify influential outliers was:

$$\text{Cook's D} > F(.5, p, n-p) \qquad (10)$$

Function Point Analysis. The BMO data base was the only data base used to investigate the "degree of correlation between 'function points' and . . . 'SLOC'" because it was the only data base that contained the required function point inputs (number of inputs, interfaces, and generated outputs) (1:639). First, using Behren's formula (Eq 4 found in Chapter II), the number of function points for each of the seven programs was calculated. Then, the ability of Albrecht and Gaffney's formulae (Eqs (5), (6), and (7), also found in Chapter II) to relate the number of function points to the number of SLOC was examined. Finally, an effort was made to derive an equation (a data base specific equation) relating the number of function points to the number of SLOC using linear regression analysis.

IV. Analysis and Results

Overview

This chapter presents the results of building a software sizing relationship for each of the four data bases and the results of the function point analysis. First, sizing equations are developed and evaluated according to the methodology discussed in Chapter III. The statistical results of this evaluation will be presented in the following order: the Ballistic Missile Office (BMO) data base, the Armament Division (AD) data base -- ground systems, the AD data base -- airborne systems, and finally the Electronic Systems Division (ESD) data base (all data bases are listed in Appendix A). Then, the results of the investigation into the correlation between function points and the number of SLOC for the BMO data base will be presented.

Development of the Sizing Equations

Development of the software sizing equations began with an analysis of each size driver. That is, each size driver (or independent variable) was plotted against the number of SLOC (dependent variable) to reveal any obvious statistical relationships (these relationships had to make logical sense before they were accepted). Then, equations that included the significant variables were analyzed.

The BMO Data Base. As described previously, the BMO data base contains seven programs described by the variables

source lines of code (SLOC), environment (ENV), language type (LANG), the number of interfaces (INTF), the number of program inputs (INPT), the number of program generated outputs (OUTPT), programmer experience level (EXP), and the number of development months (DM).

Although the data set only contains seven observations, it was decided to delete (for the regression analysis) observation #4 from the test data base because of the size of this observation's program (number of SLOC: 112,000). That is, observation #4 was not considered part of the population represented by the other six observations. Moreover, while Whetstone's study used development months (DM) as a predictor of SLOC, this study did not consider DM because most software cost models use SLOC as a predictor of development time. One good example is Boehm's Intermediate COCOMO (Constructive Cost Model), nominal effort equation for a semidetached mode program:

$$(MM)nom = 3.2[(KDSI)^{1.05}] \qquad (11)$$

where

MM = the number of man-months
KDSI = the number of thousands of delivered source instructions (4:57,117)

Thus, the six remaining observations' independent variables were plotted against SLOC in an attempt to reveal any obvious statistical relationships. After trying several logical transformations (such as square root), it was found

that a logarithmic transformation of INTF, INPT, and OUTPT best fit the data (INTFLOG, INPTLOG, and OUTPTLOG). This type of transformation makes logical sense for these variables because it means that as the number of interfaces, inputs, and outputs increase, the number of SLOC will also increase, but at a decreasing rate.

An exponential transformation (square root) of EXP was found to best fit the data. This transformation (EXPSQRT) says that as the experience level of programmers increases, the number of SLOC increase, but at a decreasing rate. Expectations would dictate that as a programmer's experience level increases, the number of SLOC needed to do a given job would decrease. However, experienced programmers, while more efficient in their coding abilities, tend to incorporate more exception handling and error detection routines in their code. Thus, this could actually increase the overall amount of code written for a development effort.

Linear forms of LANG and ENV (with negative slopes) were found to be the best for these two independent variables. While the ARINC rankings for LANG do not allow general interpretation of the effect of language, in this case such interpretation is possible since the BMO data base programs are coded in assembly language or Jovial. For this data set, a negative slope makes logical sense for LANG, because the higher-order the language, the less lines of code needed to get the job done. However, a negative slope does not seem to

make sense for ENV. Normally, software applications for space-based systems are more complex (thus, larger in size) than those for ground-based systems. Thus, the independent variable ENV was not considered in the regression analysis. The following table summarizes the results of the individual runs for each of the variables previously described (each was run against SLOC, the dependent variable).

TABLE III

Results of Individual Regression Models for SLOC
Using INTFLOG, INPTLOG, OUTPTLOG, EXPSQRT, and LANG

INTFLOG:		EXPSQRT:	
F value:	1.6930	F value:	6.2820
R^2:	0.2974	R^2:	0.6110
t(calc):	1.3010	t(calc):	2.5060
INPTLOG:		LANG:	
F value:	0.1240	F value:	2.6350
R^2:	0.0396	R^2:	0.3971
t(calc):	-0.3520	t(calc):	-1.6230
OUTPTLOG:			
F value:	0.2420	F(.90;1,4)	= 4.540
R^2:	0.0748	t(.80;4)	= 1.533
t(calc):	0.4920		

The only two significant variables were EXPSQRT and LANG (both variables' |t(calc)| > table t = 1.533). However, all five variables were included in a stepwise (backward) regression run because, as Neter points out,

> . . . each of the estimated regression coefficients individually may be statistically not significant

even though a definite statistical relation exists between the dependent variable and the set of independent variables [22:385].

Although the default level of confidence in SAS for a backward-stepwise regression run is .90, this study specified a .80 level of confidence. The results of the stepwise regression revealed a model that included INTFLOG, INPTLOG, and OUTPTLOG (see Table IV). While the model's R^2 is .9393 and the CV is relatively low (41.6%), the variable INTFLOG and the overall model are insignificant. Moreover, the negative coefficient for INPTLOG does not make sense.

TABLE IV

Results of Regression Model for SLOC
Using INTFLOG/INPTLOG/OUTPTLOG

Model: SLOC = -29942.1 + 8463.9INTFLOG - 22542.3INPTLOG + 27875.2OUTPTLOG

F value:	5.1610	
R^2:	0.9393	F(.90;3,2) = 9.160
ADJ R^2:	0.7573	t(.80;2) = 1.886
CV:	41.6234	
t(calc):		
INTFLOG:	0.6850	
INPTLOG:	-3.5410	
OUTPTLOG:	2.9930	

That is, holding all else constant, an increase in the number of program required inputs means a decrease in the number of SLOC needed. Finally, because the data base contains only six observations and there are four parameters in this model

(the three size drivers and the Y-intercept term), degrees of freedom became crucial.

Thus, it was decided to run a regression model for INPTLOG and OUTPTLOG (dropped the insignificant variable INTFLOG). The following table contains the statistical results of this model.

TABLE V

Results of Regression Model for SLOC
Using INPTLOG/OUTPTLOG

Model: SLOC = -17000.4 - 23287.8INPTLOG + 30942OUTPTLOG

F value: 10.2230
R^2: 0.9109
ADJ R^2: 0.8218
CV: 35.6689

t(calc):
INPTLOG: -4.3320
OUTPTLOG: 4.4220

F(.90;2,3) = 5.460
t(.80;3) = 1.638

VIF:
INPTLOG: 4.3425
OUTPTLOG: 4.3425

TOLERANCE:
INPTLOG: 0.2303
OUTPTLOG: 0.2303

Outliers w.r.t. X:
Leverage = 2p/n = 6/6 = 1

no observations were found
to be outliers w.r.t. X

Influential Outliers:
F distribution:
F(.50;2,3) = .88100
COOK'S D:
OBS #1: 6.75880
OBS #6: 1.57550

Outliers w.r.t. Y:
t distribution:
t(.80;3) = 1.6380
SRESID:
no observations found

As seen in the table, both variables and the model are significant. The model explains 91% of the variation in SLOC, while the adjusted R^2 of .8218 means that each var-

iable explains different sizeable "chunks" of the dependent variable. Moreover, the model's coefficient of variation (CV) is a respectable 35.7% (when predicting from the middle of the data set, this model's estimate will be accurate within plus or minus 71.4%).

Collinearity is not a problem within the model. Both variables' VIFs are less than ten and the tolerances are greater than .10. Moreover, while no observations were found to be outliers w.r.t. X or Y, observations #1 and #6 were revealed to be influential outliers (this situation is possible if a variable's SRESID and leverage values are simultaneously large). Although these two observations were influential outliers, both were kept in the data base because the observations are considered members of the population represented by the data base, and the size of the data base is critical. Finally, caution must be exercised if the model were to be used because of INPTLOG's negative coefficient (which was discussed earlier).

The AD Data Base -- Ground Systems. This data base of 12 observations contains six functional variables in addition to the number of SLOC. They are: development months (DM), language type (LANG), quality of program specification (QSPEC), reliability requirement of the software (REL), the program's (or software's) function (FUNC), and the program's complexity factor (COMPX).

As before, the functional (or independent) variable DM was not considered because number of SLOC is considered a predictor of DM. The five remaining variables were each plotted against SLOC to help reveal any statistical relationships. The only obvious transformation was an exponential transformation (COMPX squared -- CPXSQRD) of the program's complexity level. The transformation CPXSQRD says that as the complexity of a software project increases, the number of SLOC will also increase (and at an increasing rate, so long as B(i) > 0). The results of the individual regression runs are shown in the following table.

TABLE VI

Results of Individual Regression Models for SLOC
Using CPXSQRD, LANG, QSPEC, REL, and FUNC

CPXSQRD:		REL:	
F value:	9.8110	F value:	0.2690
R^2:	0.4952	R^2:	0.0262
t(calc):	3.1320	t(calc):	-0.5190
LANG:		FUNC:	
F value:	0.2470	F value:	1.4920
R^2:	0.0241	R^2:	0.1298
t(calc):	-0.4970	t(calc):	-1.2210
QSPEC:			
F value:	0.2520	F(.90;1,10)	= 3.290
R^2:	0.0246	t(.80;10)	= 1.372
t(calc):	-0.5020		

As shown in the table, the only significant variable was CPXSQRD (when regressed individually against SLOC). The major reason why only a few variables in each of these data bases were significant is that the small number of observations did not provide the necessary "variability" in the data. That is, the values for each of the size drivers do not provide enough "difference" to generate a statistical relation. An excellent example from this data base is the program's quality of specification (or QSPEC). A look at the data base reveals nine two-values, two three-values, and one one-value observation for QSPEC. Thus, QSPEC was revealed to be a poor size driver for this data base.

Regressing the five variables in a backward-stepwise SAS run revealed a significant model that included the variables CPXSQRD and FUNC. Table VII summarizes the results of the regression run for CPXSQRD and FUNC.

Although the model and both variables are significant, the model is only a fair predictor of SLOC. When predicting from the middle of the data, this model's estimate will be accurate within plus or minus 145%. This is one indication that an important size driver is missing from the model.

An interesting point is that although FUNC was not individually significant, it was significant when included in a model with CPXSQRD. As explained earlier, it is possible that the regression coefficients individually may

be statistically insignificant although a statistical relationship exists between the dependent and the set of independent variables (22:385).

TABLE VII

Results of Regression Model for SLOC
Using CPXSQRD/FUNC

Model: SLOC = 28016.2 + 2418.8CPXSQRD - 18302.1FUNC

F value: 7.6270
R^2: 0.6289
ADJ R^2: 0.5465
CV: 77.4000

t(calc):
CPXSQRD: 3.4790
FUNC: -1.8010

F(.90;2,9) = 3.010
t(.80;9) = 1.383

VIF:
CPXSQRD: .99994
FUNC: .99994

TOLERANCE:
CPXSQRD: 1.00005
FUNC: 1.00005

Outliers w.r.t. X:
Leverage = 2p/n = 6/12 = .50

no observations were found
to be outliers w.r.t. X

Influential Outliers:
F distribution:
F(.50;2,9) = .74900
COOK'S D:
OBS #1: 1.14409

Outliers w.r.t. Y:
t distribution:
t(.80;9) = 1.3830
SRESID:
OBS #1: 2.3491
OBS #11: -1.7035

Moreover, a negative coefficient for FUNC logically makes sense because as the function of software moves from missile applications to munitions (from more complex to less complex), the expectation would be a decrease in the number of SLOC required for the application.

A closer look at the model statistics shows that while the two variables are almost perfectly independent, they reduce the variation in SLOC by only 63%. However, an adjusted R^2 of .55 means that each variable explains different "chunks" of the dependent variable. Moreover, while there were no outliers w.r.t. X, observations #1 and #11 were found to be outliers w.r.t. Y (observation #1 was also found to be an influential outlier). Both observations were considered members of the data base's population. Thus, the observations were kept.

The AD Data Base -- Airborne Systems. This Armament Division data base of 13 observations contains the same six functional variables (in addition to the number of SLOC) that the ground systems data base contains. Again, the functional variable DM was not considered.

Plotting the remaining functional variables against SLOC, and trying some logical transformations, revealed exponential transformations for the program's complexity (COMPX squared -- CPXSQRD) and reliability (REL squared -- RELSQRD), and an inverse transformation for the program's function (1/FUNC squared -- FUNC2INV). The transformation for the program's function means that as the function of the software product goes from missile systems to munitions (thus, decreasing in complexity), the expectation is a decrease in the lines of code required, but at a decreasing rate (the squared transformation was explained earlier).

Table VIII reflects the results of the individual regression runs.

TABLE VIII

Results of Individual Regression Models for SLOC
Using CPXSQRD, RELSQRD, FUNC2INV, LANG, and QSPEC

CPXSQRD:		LANG:	
F value:	10.4380	F value:	0.0400
R^2:	0.4869	R^2:	0.0036
t(calc):	3.2310	t(calc):	-0.2000
RELSQRD:		QSPEC:	
F value:	6.1220	F value:	0.5630
R^2:	0.3575	R^2:	0.0487
t(calc):	2.4740	t(calc):	-0.7510
FUNC2INV:			
F value:	3.8440	F(.90;1,11)	= 3.230
R^2:	0.2590	t(.80;11)	= 1.363
t(calc):	1.9610		

Although only three of the five variables were individually significant (CPXSQRD, RELSQRD, and FUNC2INV), all five were included in the stepwise regression run. The backward elimination of variables resulted in the two-variable model, CPXSQRD and QSPEC (see Table IX for the model's regression statistics).

Although this model's CV is relatively high (65.9%), the model is significant beyond the 99% level of confidence, and it explains more than two-thirds of the variation in SLOC. However, while collinearity within the model is not a

problem, outliers w.r.t. X and Y were identified. Observations #1 and #6 were found to be outliers w.r.t. X, and observations #6 and #8 were found to be outliers w.r.t. Y (observation #6 was also the influential outlier). Since the three observations were considered members of the population represented by the data base, they were kept.

TABLE IX

Results of Regression Model for SLOC Using CPXSQRD/QSPEC

Model: SLOC = 54964.6 + 1393.1CPXSQRD - 28930.7QSPEC

F value: 11.4910
R^2: 0.6968
ADJ R^2: 0.6362
CV: 65.9447

t(calc):
CPXSQRD: 4.6230
QSPEC: -2.6310

F(.90;2,10) = 2.920
t(.80;10) = 1.372

VIF:
CPXSQRD: 1.10500
QSPEC: 1.10500

TOLERANCE:
CPXSQRD: 0.90490
QSPEC: 0.90490

Outliers w.r.t. X:
Leverage = 2p/n = 6/13 = .4615
OBS #1: 1.00000
OBS #6: 0.52843

Influential Outliers:
F distribution:
F(.50;2,10) = 0.74300
COOK'S D:
OBS #6: 1.27419

Outliers w.r.t. Y:
t distribution:
t(.80;10) =1.3720
SRESID:
OBS #6: 1.8469
OBS #8: -1.7693

Finally, the negative coefficient for the program's quality of specification (QSPEC) makes sense because, as a program's QSPEC increases (becomes more well defined), the size of the program should decrease.

The ESD Data Base. The 97 observations that make up this data base are described by four functional variables in addition to the number of delivered source lines of code (DSLOC). They are: the program's complexity (CPLX), the program's required reliability (RELY), the level of modern programming practices used in coding the program (MODP), and the level of software tools used (TOOL).

Plotting each of the functional (or independent) variables against DSLOC revealed very little. Therefore, forms of two (RELY and CPLX) of the functional variables that were encountered in previous data sets were used for the individual regression runs. The results of these runs are summarized in Table X.

As indicated by the table, none of the variables proved statistically significant. However, a backward-stepwise regression was conducted using the six variables identified in Table X. As seen in table XI, the result of the stepwise regression run was a two-variable model containing MODP and TOOL. Although the variables are significant, the overall model is insignificant. Moreover, the model only explains about 6% of the variation in SLOC, and the coefficient of variation is extremely high (143.88).

TABLE X

Results of Individual Regression Models for SLOC
Using CPLX, CPLXSQRD, RELY, RELYSQRD, MODP, and TOOL

CPLX:		RELYSQRD:	
F value:	0.1100	F value:	0.4310
R^2:	0.0012	R^2:	0.0045
t(calc):	0.3310	t(calc):	-0.6570
CPLXSQRD:		MODP:	
F value:	0.1170	F value:	1.1190
R^2:	0.0012	R^2:	0.0149
t(calc):	0.3410	t(calc):	-1.0580
RELY:		TOOL:	
F value:	0.2650	F value:	0.0830
R^2:	0.0028	R^2:	0.0009
t(calc):	-0.5150	t(calc):	0.2890

F(.90;1,95) = 2.770
t(.80;95) = 1.292

With the number of observations contained in the data base, one would expect to be able to better explain the dependent variable. However, an inspection of the data reveals the same problem encountered previously -- the independent variables do not provide enough difference (or variability) to generate a significant statistical relationship. Thus, a significant regression model could not be developed for this data base.

TABLE XI

Results of Regression Model for SLOC Using MODP/TOOL

Model: SLOC = 35902.1 - 20675.7MODP + 26616.9TOOL

F value:	2.2260	
R^2:	0.0575	F(.90;2,94) = 2.370
ADJ R^2:	0.0317	t(.80,94) = 1.292
CV:	143.8834	

t(calc):

MODP:	-1.7140
TOOL:	1.8160

Function Point Analysis

The final portion of this study's analysis investigated the degree of correlation between function points (as defined by Albrecht and Gaffney) and SLOC. The BMO data base was the only data base of the four used in this investigation because it was the only one that contained the required function point inputs (or variables) -- the number of inputs, interfaces, and program generated outputs.

First, the number of function points for each of the seven programs using Behren's formula was calculated. Using observation #1 from the data base, the function point count for this observation was calculated in the following manner: 4(10) + 5(37) +4(0) + 10(0) + 7(14) = 323. Then, the function point counts as calculated by Behren's formula were input into Eqs (5), (6), and (7) of Chapter II to determine the predicted number of source lines of code. Using observation #1 again, the predicted SLOC (using

Eq (7)) were calculated in the following manner: 53.2(323) + 12773 = 29957. Table XII summarizes the results for these calculations.

TABLE XII

Comparison of Predicted and Actual SLOC Counts Generated by Albrecht and Gaffney's Formulae

OBS	Function Point Count	Predicted SLOC Using Eq (5)	Predicted SLOC Using Eq (6)	Predicted SLOC Using Eq (7)	Actual SLOC
1	323	31850	19011	29957	43000
2	144	10603	05926	20434	08875
3	147	10959	06146	20593	32000
4	451	47044	28807	36766	112000
5	270	25559	15137	27137	06400
6	376	38141	22886	32776	13010
7	1337	152212	93135	83901	16000

A closer inspection of Table XII reveals that Albrecht and Gaffney's formulae do a poor job of relating function points to the number of SLOC for this data base. There are two possible explanations for this. First, Albrecht and Gaffney's study was based upon COBOL and PL/1 programs.

However, the programs in the BMO data base are coded in assembly, Jovial, or event driven languages. Second, Albrecht and Gaffney's study was conducted using business-application programs, whereas, the BMO data base contains real-time ballistic missile applications.

Finally, since Albrecht and Gaffney's formulae are data base specific (as shown in Table XII above), it was decided to attempt to develop (through linear regression analysis) a data base specific equation for the BMO data that relates function points (FPNT) to the number of SLOC. As shown in Table XIII, a significant relationship could not be developed.

TABLE XIII

Results of Regression Model for FPNT

Model: SLOC = 34023.7 - 2.2575FPNT

F value:	0.0030	
R^2:	0.0006	F(.90;1,5) = 4.060
ADJ R^2:	-0.1992	t(.80;5) = 1.476
CV:	123.4000	
t(calc):		
EXPSQRT:	-0.0560	

V. Conclusions and Recommendations

Conclusions

This study's first research question posed in Chapter I was -- Using size drivers identified in the literature and identified by Whetstone's study, how well can regression analysis relate software size to these size drivers? Chapter IV answered this question for each of the four data bases considered (Table XIV contains a summary of results of the regression analyses).

For the BMO data base, the regression analysis revealed that the square root of the programmers' experience level (EXPSQRT) and language type (LANG) were the individually significant variables. However, all five variables were included in a backward-stepwise regression run. The model identified by this regression run as the best predictor of SLOC was a two-variable model containing INPTLOG and OUTPTLOG. Both the model and the variables were significant well beyond this study's level of confidence constraints. Moreover, the coefficient of correlation (R^2) for this model was .91. However, the model's coefficient of variation (CV = 35.7) signals the model as only a fair predictor of number of SLOC, and the negative coefficient for INPTLOG logically does not make sense.

The AD data base was divided into two subgroups according to operational environment -- ground programs and airborne programs. As Whetstone notes, "this was done in

TABLE XIV

Summary of Results

Data Base	Model	R^2
BMO	SLOC = -17000.4 - 23287.8INPTLOG + 30942OUTPTLOG	.9109
AD Ground	SLOC = 28016.2 + 2418.8CPXSQRD - 18302.1FUNC	.6289
AD Airborne	SLOC = 54964.6 + 1393.1CPXSQRD - 28930.7QSPEC	.6968
ESD	None	N/A

order to better separate the different types of software" (32:83). The results of the ground data base indicated that only the square of the program's complexity (CPXSQRD) was significant in an individual regression model. However, when all five variables were included in a backward-stepwise regression run, the two-variable model, CPXSQRD and FUNC, was the result.

The regression analysis performed on the airborne AD data base revealed that the square of the program's complexity (CPXSQRD), the square of the program's required reliability (RELSQRD), and the inverse of the program's function, squared (FUNC2INV) were the statistically significant variables. However, a stepwise (backward) regression run that included all the variables revealed

that the statistically significant model for this data base contained the variables CPXSQRD and QSPEC. Although the model explains over two-thirds of the variation in the dependent variable, the model's CV is relatively high (65.9%).

Lastly, the regression analysis performed on the ESD data base revealed no significant variables. Because only four independent variables were investigated, a statistically significant model could not be found for the data. Although this data base contains 97 observations, the lack of difference (or variability) in the values of the 97 data points makes it almost impossible to generate significant statistical relationships. Thus, an investigation of how ESD's data was gathered/encoded is needed.

Overall, the best predictive model found was for the BMO data base. The R^2 for this model is 91.9% and the adjusted R^2 is 82.2%. However, the models that were created for each data base are not very helpful in predicting the number of SLOC. The main reasons for this are the relatively low R^2 values and the high CV values which indicate that other independent variables related to SLOC are missing from the models or that there is just a lot of variability in SLOC for a program. Moreover, the small number of data points encountered for the BMO data base and the two AD data bases cause any regression results to be somewhat suspect.

Finally, the results of the investigation into the degree of correlation between function points and software size were also presented in Chapter IV. No correlation between function points and software size (SLOC) was found.

Recommendations

The results of this study have generated a few suggestions for further research.

Recommendation 1. A study investigating Air Force software data collection methods is needed. From all the research conducted by the author, software costing and sizing historical data is lacking. Because of software's increasing importance, this type of study could be extremely worthwhile.

Recommendation 2. Larger/validated data bases with more independent variables are needed before any further research is accomplished (this would probably make the results more substantial and meaningful). Once these data bases have been located/formed, multiplicative models or nonlinear regression techniques may produce better results.

Recommendation 3. Further study is needed to investigate the application of Albrecht and Gaffney's function point methodology to Air Force programs. Because of the real-time nature of most Air Force software products, this investigation should include a consideration of Reifer's ASSET-R sizing model.

Appendix A: Thesis Data Bases

I. Ballistic Missile Office

OBS	LOC	ENV	LANG	INTF	INPT	OUTPT	EXP	DM
1	43000	1	3	14	10	37	24	36
2	8875	2	3	10	6	10	8	30
3	32000	1	3	21	--	--	54	30
4	112000	1	5	9	42	44	1	26
5	6400	1	5	7	24	25	1	26
6	13010	2	5	16	31	28	8	31
7	16000	1	4.5	18	139	131	16	29

REPRINTED FROM: (32:105)

II. Armament Division -- Ground

OBS	LOC	DM	LANG	QSPEC	REL	FUNC	COMPX
1	80000	48	1.3	2	3	2	5
2	25000	48	1.3	2	3	2	3
3	10000	20	1.2	2	5	2	3
4	4000	22	1.2	2	4	2	3
5	3000	24	1	1	2	2	2
6	1400	36	8	3	4	3	4
7	15900	36	1	2	3	3	4
8	1200	36	1	3	3	3	3
9	25000	--	1.3	2	3	2	3
10	40000	--	1	2	3	2	5
11	5800	--	3	2	3	2	4
12	25000	--	8	2	4	2	4

REPRINTED FROM: (32:108)

III. Armament Division -- Airborne

OBS	LOC	DM	LANG	QSPEC	REL	FUNC	COMPX
1	3000	24	2.05	3	4	1	5
2	2100	18	3	2	1	1	1
3	2000	42	3	2	3	3	2
4	16000	36	3	2	4	1	4
5	30000	24	1.3	2	5	1	4
6	60000	79	3	2	5	1	6
7	1300	22	1.2	2	3	2	3
8	16000	24	1	2	5	1	5
9	2700	22	1	2	4	2	3
10	10000	--	3	2	3	2	1
11	9000	--	6.8	2	5	2	4
12	17000	--	1	2	4	1	4
13	29000	--	1	2	4	1	4

REPRINTED FROM: (32:109)

IV. Electronic Systems Division

OBS	DSLOC	CPLX	RELY	MODP	TOOL
1	26200	5	5	4	3
2	15987	4	4	4	1
3	56021	4	5	4	4
4	20276	4	4	3	3
5	63944	5	4	3	1
6	47525	5	4	4	4
7	09000	5	4	1	3
8	54192	5	5	4	4
9	104090	5	5	4	4
10	71453	5	5	4	4
11	75081	6	5	4	4
12	40702	5	5	-	4
13	21502	5	5	-	4
14	14817	5	5	-	4
15	14809	4	5	-	4
16	12774	5	5	-	4
17	14839	5	5	-	4
18	29445	5	5	-	4
19	26933	5	5	-	4
20	20547	3	3	-	4
21	23664	3	3	-	4
22	00411	4	3	-	4
23	25212	4	3	-	4
24	10991	3	5	-	4

IV. Electronic Systems Division (Continued)

OBS	DSLOC	CPLX	RELY	MODP	TOOL
25	06450	3	5	-	4
26	32100	5	5	-	4
27	01508	3	3	-	4
28	15000	4	3	-	4
29	11000	2	4	-	4
30	01700	2	4	-	4
31	37000	4	3	-	4
32	30000	5	5	-	4
33	221990	4	5	4	5
34	116098	3	3	4	5
35	250739	4	5	5	4
36	60031	3	5	3	3
37	01280	3	5	3	3
38	09445	3	5	3	3
39	32467	3	5	3	3
40	04072	3	5	3	3
41	20901	3	5	3	3
42	11986	3	5	3	3
43	11696	3	5	3	3
44	04994	3	5	3	3
45	02035	3	5	3	3
46	08500	5	5	4	4
47	11691	5	5	4	4
48	47723	3	5	3	3

IV. Electronic Systems Division (Continued)

OBS	DSLOC	CPLX	RELY	MODP	TOOL
49	02704	3	5	3	3
50	03610	3	5	3	3
51	16328	3	5	3	3
52	14529	3	5	3	3
53	293000	5	4	2	3
54	82000	3	3	2	3
55	246000	3	3	2	3
56	492000	3	3	2	3
57	11239	4	2	3	3
58	11935	4	2	3	3
59	68922	4	2	3	3
60	05639	4	2	3	3
61	07222	4	2	3	3
62	18751	4	2	3	3
63	101800	1	3	5	3
64	67200	4	3	5	3
65	54958	5	5	2	3
66	20782	4	5	2	3
67	41299	4	5	2	3
68	46420	3	3	2	3
69	02801	3	3	4	4
70	12000	5	4	4	3
71	106000	4	4	4	3
72	09200	3	3	4	3

IV. Electronic Systems Division (Continued)

OBS	DSLOC	CPLX	RELY	MODP	TOOL
73	16354	3	3	4	3
74	11700	3	3	4	3
75	21352	3	2	4	3
76	41470	3	4	3	3
77	61791	2	3	3	3
78	28960	3	3	4	3
79	54615	3	3	4	3
80	43907	4	3	4	3
81	09601	2	3	4	3
82	14177	3	3	4	3
83	02695	2	3	4	4
84	14531	3	3	4	3
85	74820	5	5	4	4
86	11000	5	5	4	4
87	62319	4	4	3	3
88	31795	4	4	3	3
89	11500	6	4	4	4
90	71600	4	4	4	4
91	21700	4	4	4	4
92	26100	3	3	4	4
93	73100	3	3	4	4
94	87700	5	4	4	4
95	18300	4	4	4	4
96	29500	4	4	4	4

IV. Electronic Systems Division (Continued)

OBS	DSLOC	CPLX	RELY	MODP	TOOL
97	50100	2	2	4	4

Bibliography

1. Albrecht, Allan J., and John E. Gaffney, Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, Vol. SE-9, No. 6: 639-648 (November 1983).

2. Analytic Sciences Corporation, The. The AFSC Cost Estimating Handbook (Volume I). Reading MA: prepared for USAF, Air Force Systems Command (AFSC), 1986.

3. Boehm, Barry W. "Software Engineering Economics," IEEE Transactions on Software Engineering, Vol. SE-10, No. 1: 4-21 (January 1984).

4. -----. Software Engineering Economics. Englewood Cliffs NJ: Prentice-Hall, Inc., 1981.

5. -----. "Understanding and Controlling Software Costs," Journal of Parametrics, Vol. VIII, No. 1: 32-68 (March 1988).

6. Dekker, M. and F.J. Bosch. Functional Requirements for a Software Cost Database. Amsterdam, the Netherlands: National Aerospace Laboratory (NLR), November 1981 (AD-B064365).

7. Ferens, Daniel V. "Computer Software Size Estimation," Proceedings of the 1988 National Aerospace and Electronics Conference, 1-6. Dayton OH: 1988.

8. -----. An Introduction to Software Parametric Cost Estimating. Class text distributed in Cost 672, Model Diagnostics/Software Management. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1987.

9. Fox, Joseph M. Software and its Development. Englewood Cliffs NJ: Prentice-Hall, Inc., 1982.

10. Funch, Paul G. Software Cost Data Base. Contract F1962886C0001. Bedford MA: The MITRE Corporation, October 1987.

11. Galorath, Daniel D. Software Engineering Project Management. Marina del Rey CA: Computer Economics, Inc., 1986.

12. Garvey, Paul R. and Frederic D. Powell. "Three Methods for Quantifying Software Development Effort Uncertainty," Journal of Parametrics, Vol. VIII, No. 1: 76-92 (March 1988).

13. Graver, C. A. et al. Cost Reporting Elements and Activity Cost Tradeoffs for Defense System Software (Volume II): Final Report, 11 March 1976 - 11 March 1977. Contract F1962876C0180. Santa Barbara CA: General Research Corporation, May 1977 (AD-A053021).

14. Gross, Stephen et al. Software Sizing and Cost Estimation Study. Report presented at the 19th Annual Department of Defense Cost Analysis Symposium. Xerox Training Center, Leesburg VA, 17-20 September 1985.

15. Itakura, Minoru, and Akio Takayanagi. "A Model for Estimating Program Size and Its Evaluation," Proceedings, Sixth International Conference on Software Engineering. 104-109. Long Beach CA: IEEE Computer Society Press, 1982.

16. Jensen, Randall W. An Improved Software Development Resource Estimation Model. Prepared for the Hughes Aircraft Company, Space and Communications Group, Los Angeles CA, n.d.

17. Jensen, Randall W. and Suzanne Lucas. Sensitivity Analysis of the Jensen Software Model. Prepared for the Hughes Aircraft Company, Space and Communications Group, Los Angeles CA, n.d.

18. Kimhan, Capt Sidney C. III and Capt David M. King. The Effects of Software Quality Control and Baseline Management on the Acquisition of Computer Programs. MS thesis, AFIT/GLM/LSM/84S-35. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1984 (AD-A147051).

19. Marks, Kenneth E. Estimating Avionics Software Development Cost: Interim Report, March 1981. Contract F4962077C0023. Santa Monica CA: The Rand Corporation, March 1981 (AD-B056716L).

20. Martin, Edith W. "Strategy for a DoD Software Initiative", Computer, 16: 52-59 (March, 1983).

21. Najberg, Andrew. "Software Size Estimation by Analogy," National Estimating Society (NES), New England Chapter, Software Cost Estimating Symposium, 20 November 1985.

22. Neter, John et al. Applied Linear Regression Models. Homewood IL: Richard D. Irwin, Inc., 1983.

23. Park, In Kyoung. Software Cost Estimation Through Bayesian Inference of Software Size. MS thesis. Naval Postgraduate School, Monterey CA, September 1985 (AD-A161184).

24. Putnam, Lawrence H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, Vol. SE-4, No. 4: 345-361 (July 1978).

25. Rather, Elizabeth D. "Controlling the Escalating Costs of Software Development," Defense Science 2001+, 2: 37-43 (June 1983).

26. Reese, Richard M. and Jim Tamulevicz. "A Survey of Software Sizing Methodologies and Tools," Journal of Parametrics, Vol. VII, No. 2: 35-55 (June 1987).

27. Reifer, Donald J. Analytical Size Estimation Tool -- Real-Time (ASSET-R): An Overview (RCI-TN-269). Torrance CA: Reifer Consultants, Inc., 2 May 1987.

28. -----. Predicting the Size of Real-Time Systems (RCI-TN-238A). Torrance CA: Reifer Consultants, Inc., May 1987.

29. Steig, Capt Jeffrey T. An Application of Discriminant Analysis to the Selection of Software Cost Estimating Models. MS thesis, AFIT/GSM/LSY/84S-26. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1984 (AD-A147632).

30. United States General Accounting Office. Report to the Congress of the United States. "DoD Needs to Provide More Credible Weapon Systems Cost Estimates to the Congress." Report series GAO/NSIAD-84-70. Washington: Government Printing Office, 24 May 1984.

31. Wheaton, Marilee J. "Functional Software Sizing Methodology," Journal of Parametrics, Vol. VI, No. 1: 17-23 (March 1986).

32. Whetstone, Capt Mark J. Developing Software Size Estimating Relationships Based on Functional Descriptions of the Software. MS thesis, AFIT/GSM/LSY/86S-24. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1986 (AD-A174335).

33. Yaru, Dr. Nicholas. Army Science Board (ASB) 1983 Summer Study on Acquiring Army Software. Report to the Department of the Army, Office of the Assistant Secretary. HQ USA, Washington DC, December 1983 (AD-B085044L).

Vita

Captain Frank Albanese, Jr. [redacted] in 1973 [redacted] attended Providence College. He received the degree of Bachelor of Arts in Mathematics in May 1977. After working for Travelers Insurance Company as a programmer-analyst, he entered Officer Training School (OTS) and earned a commission in the USAF in April 1981. He then entered navigator training and earned his wings in November 1981. After attending KC-135 Combat Crew Training School (CCTS) at Castle AFB, California, he was assigned to the 70th Air Refueling Squadron, 305th Air Refueling Wing at Grissom AFB, Indiana, in June 1982. While there, he served as a squadron navigator, instructor navigator, and wing current operations officer until entering the School of Systems and Logistics, Air Force Institute of Technology, in May 1987.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCA/LSY/88S-1	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Systems and Logistics 6b. OFFICE SYMBOL (If applicable) AFIT/LSQ	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) Frank Albanese, Jr., M.A., Capt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____ 14. DATE OF REPORT (Year, Month, Day) 1988 September 15. PAGE COUNT 85	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Cost Analysis, Regression Analysis, Estimates, Costs, Computer Programs
05	03		
12	03		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: THE APPLICATION OF REGRESSION-BASED AND FUNCTION POINT SOFTWARE SIZING TECHNIQUES TO AIR FORCE PROGRAMS

Thesis Advisor: Daniel V. Ferens
Associate Professor of Systems Management

Approved for public release IAW AFR 190-1.

WILLIAM A. MAUER 17 Oct 88
Associate Dean
School of Systems and Logistics
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Daniel V. Ferens	22b. TELEPHONE (Include Area Code) (513) 255-4845	22c. OFFICE SYMBOL AFIT/LSY

DD Form 1473, JUN 86 *Previous editions are obsolete.*

This study investigated the ability to develop regression models to predict the number of source lines of code (SLOC) and the degree of correlation between function points and the number of SLOC. Since software size -- or the number of lines of code (LOC) -- is considered to be the primary software cost driver, accurate software size estimates are critical because of the growing importance of software in today's Department of Defense (DoD) weapon systems.

For the regression analysis, four sizing data bases, containing various functional (independent) variables, were used. These variables included complexity, reliability, the program's quality of specification, etc. Regression analysis for each data base was performed with the goal of deriving an optimal model to predict SLOC. Linear forms and nonlinear transforms of the independent variables were used in the analysis. Of the three data bases containing complexity, the squared exponential transform of the variable was statistically significant in two. The best model found was for the Ballistic Missile Office (BMO) data base. The coefficient of correlation was .9109 for the variables INPTLOG (the log of the number of program inputs) and OUTPTLOG (the log of the number of program outputs).

For the function point analysis, the variables of number of inputs, outputs, and interfaces (from the BMO data base) were used to apply Albrecht and Gaffney's methodology to investigate the correlation of function points to software size. No correlation for this particular data base could be found. More data is needed before further research can be accomplished.